102

T_1: {a, b, c, d, e, f, g}
T_2: {d, f, g}
T_3: {a, b, d, g}
T_4: {a, d, g}
T_5: {f, g }
T_6: {e, f, g}
T_7: {e, g }

108

Example for Dependency (50% as threshold)

{ag}

$P(a|g) = count(ag)/count(g) = 3/7$

$P(g|a) = count(ag)/count(a) = 3/3$

a->g, but not g->a

{ab}

$P(a|b) = 2/2$

$P(b|a) = 2/3$

a -> b, and b->a, (ab) is not frequent

Patterns	Count
a	3
b	2
c	1
d	3
e	3
f	4
g	7

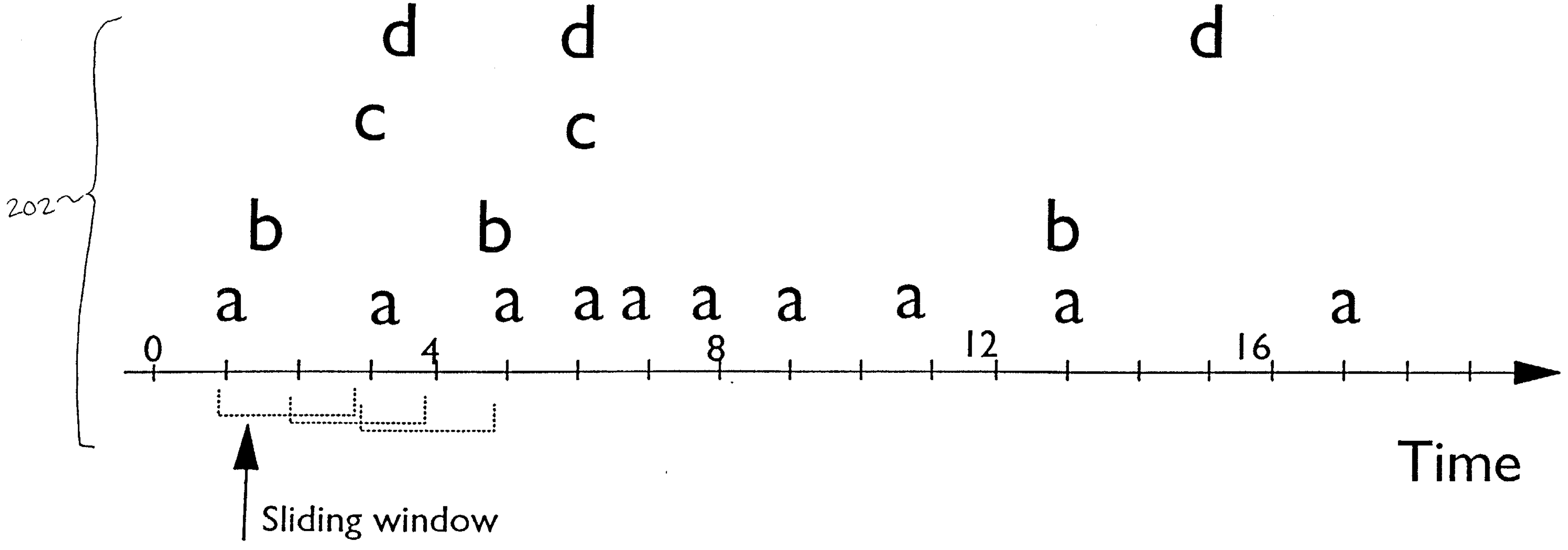
104

Patterns	Count
a b	2
a d	3
a e	1
a f	1
a g	3
......	

106

FIG. 1

YOR920000679US1



Patterns	Count
a	10
b	3
c	2
d	3

204

Patterns	Count
ab	3
ac	2
dc	2
....	

206

minsup = 3; minp = 0.6

{ab} is frequent, but not m-pattern

P(a|b) = count(ab)/count(b) = 1;

P(b|a) = 3/10

{dc} is m-pattern, but not frequent

P(d|c) = 2/3; P(c|d) = 1;

208

FIG. 2

3/10
YOR920000679US1

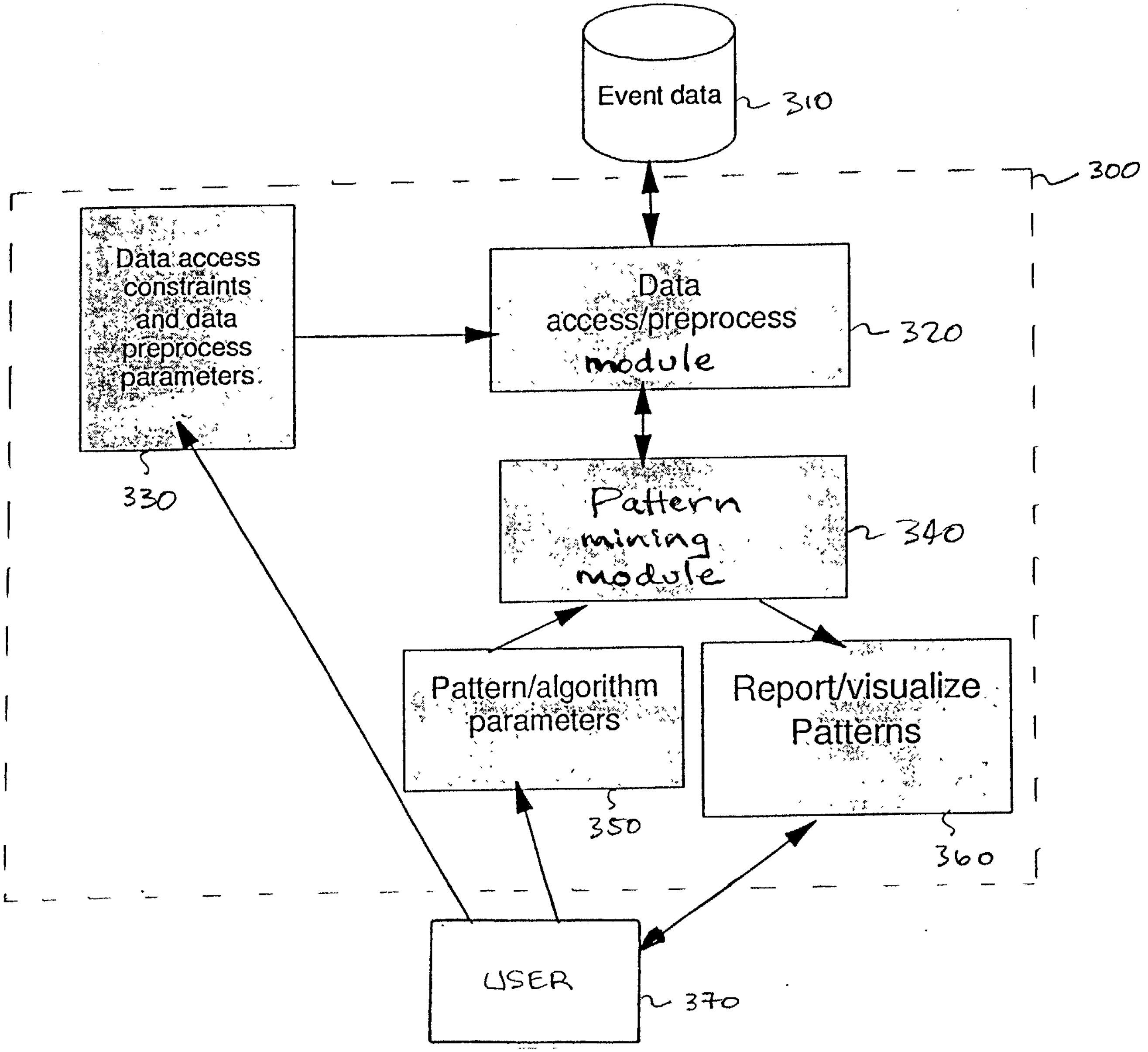
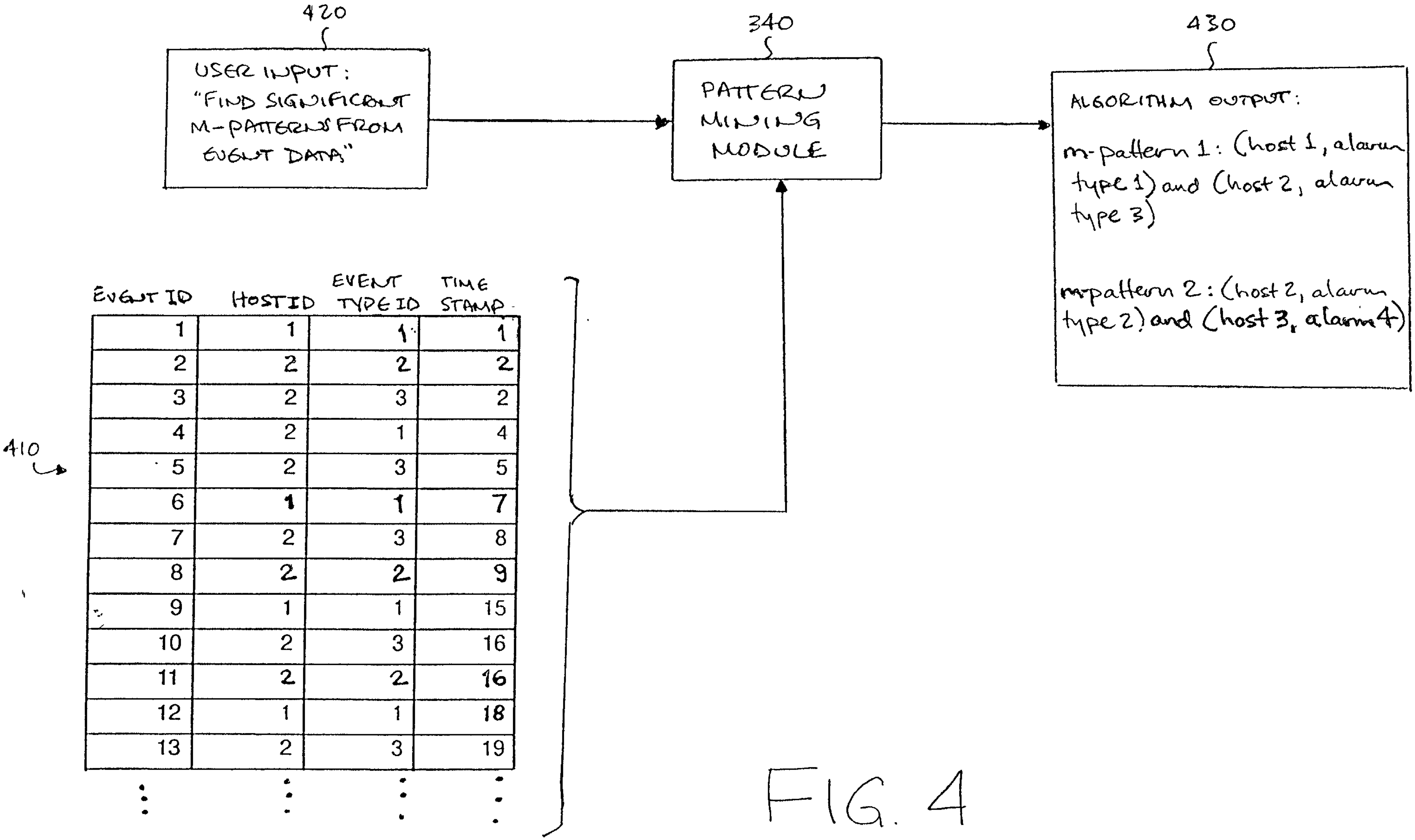


FIG. 3



EVENT ID	HOSTID	EVENT TYPE ID	TIME STAMP
1	1	1	1
2	2	2	2
3	2	3	2
4	2	1	4
5	2	3	5
6	1	1	7
7	2	3	8
8	2	2	9
9	1	1	15
10	2	3	16
11	2	2	16
12	1	1	18
13	2	3	19
⋮	⋮	⋮	⋮

FIG. 4

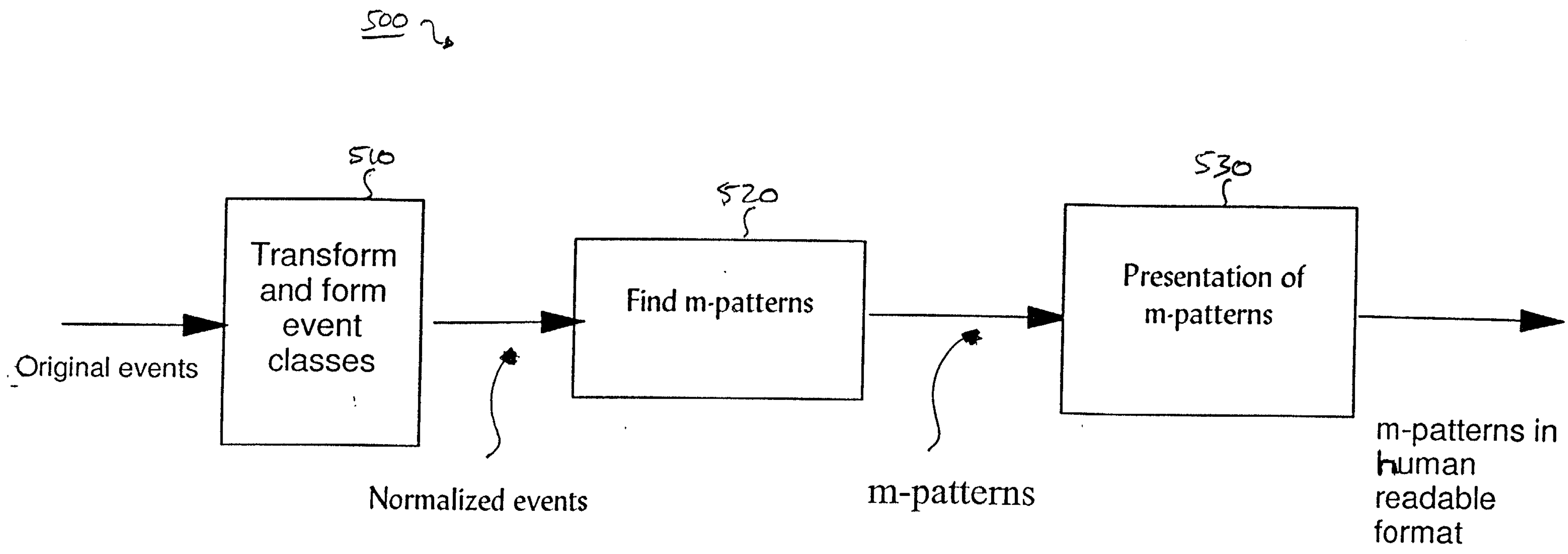

500
510
Transform and form event classes
520
Find m-patterns
530
Presentation of m-patterns
Original events
Normalized events
m-patterns
m-patterns in human readable format

FIG. 5

610

Event ID	Event type ID	Host ID	Time stamp
1	1	1	1
2	2	2	2
3	1	1	4
4	1	1	7
5	2	2	9
6	1	1	15
7	2	2	16
8	1	1	18
9	1	3	19
10	2	1	21
11	2	2	23
12	2	2	25
13	1	1	30

Table: original events

step 510

{Event type ID, host ID}	Event class
{1, 1}	1
{1, 3}	2
{2,1}	1
{2,2}	4

620

Table: mapping for event class

EVENT ID	EVENT CLASS	TIME STAMP
1	1	1
2	4	2
3	1	4
4	1	7
5	4	9
6	1	15
7	4	16
8	1	18
9	2	19
10	1	21
11	4	23
12	4	25
13	1	30

630

Table: event after mapping

FIG 6

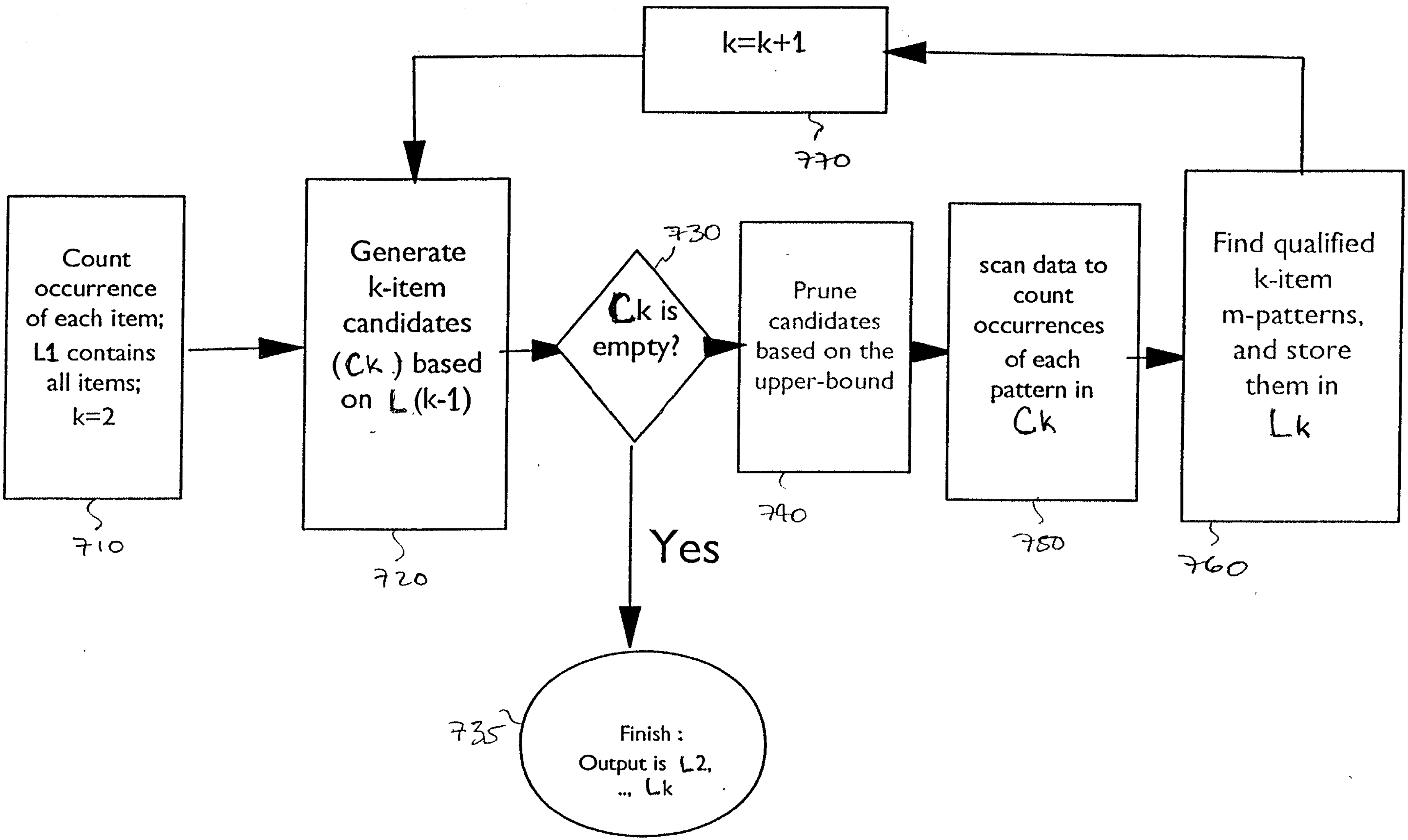

Count occurrence of each item; L1 contains all items; k=2
710
Generate k-item candidates (Ck) based on L(k-1)
720
Ck is empty?
730
Yes
Finish : Output is L2, ..., Lk
735
Prune candidates based on the upper-bound
740
scan data to count occurrences of each pattern in Ck
750
Find qualified k-item m-patterns, and store them in Lk
760
k=k+1
770

FIG 7

- Input: a set of candidates *Ck*, count information at all previous levels and a threshold minp
- Output: a set of pruned candidates *C'k*
- Algorithm
  - For each pattern *pat* in *Ck*
    - For each item *a* in *pat*
      - Compute: *prob* = *Count(pat-a)/Count(a)*;
      - if *prob* < *minp*
        - *Ck* = *Ck-pat*
        - break the for-loop
  - Return *Ck*

FIG. 8A

- Input: pattern *pat*, all count information, and a threshold *minp*
- Output: true if *pat* is a qualified m-pattern; otherwise false.
- Algorithm
  - For each *a* in *pat*
    - $prob = Count(pat)/Count(a)$
    - if $prob < minp$
      - return false
  - Return true
- This algorithm is O(k)

FIG. 8B

10/10
YOR920000679US1

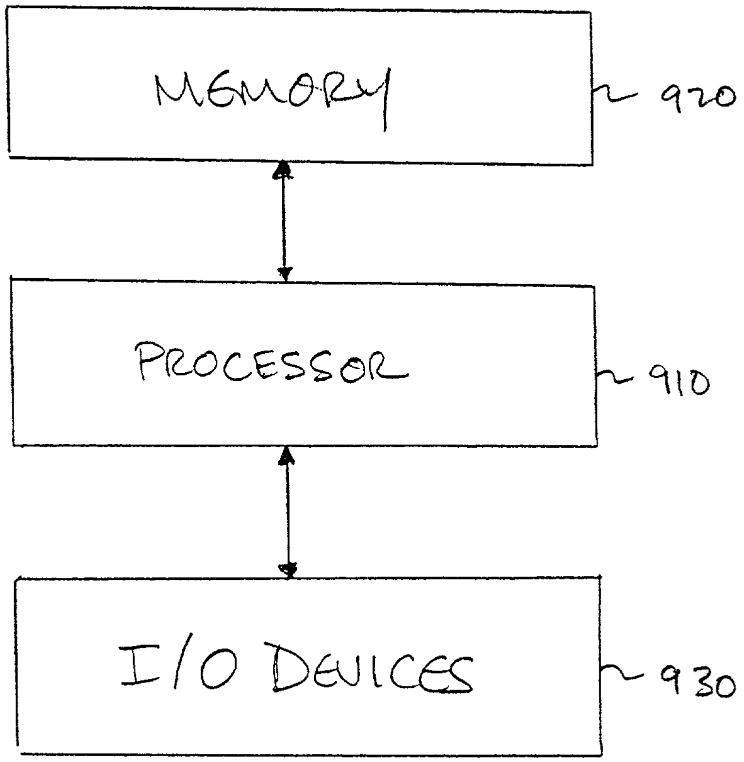


FIG. 9